Configuration Manual

MSc Research Project
MSCCYBETOP

Senan Behan
Student ID:x20167601

School of Computing
National College of Ireland

Supervisor: Mr. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Senan Behan…………………………………………………………………………

Student ID: ……x20167601…………………………………………………….…….

Programme: ……MSCCYBETOP………………… **Year:** ……2022…

Module: ……Research Project …………………………………….………

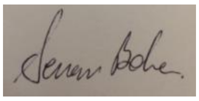
Lecturer: ……Vikas Sahni …………………………………………………….………

Submission Due Date: ……15th August 2022…………………………………………………………….………

Project Title: … Solidity Smart Contract Testing with Static Analysis Tools …

Word Count: …………1711………………… **Page Count:** ………14….………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ………Senan Behan……………………………………  ……………………

Date: …………14/08/2022……………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Senan Behan
Student ID: x20167601

1 Equipment

1.1 Equipment utilised

Compiling and organisation of selected smart contracts in to separate files was conducted utilising VS Code version: 1.59.1 (user setup), Node.js: 14.16.0, V8: 9.1.269.36-electron.0, OS: Windows_NT x64 10.0.19044. The VS Code was installed on Operating System Windows 10 Pro version 10.0.19044 Build 19044, with hardware: Lenovo Thinkpad-26 , Processor Intel® Core™ i7-6700HQ CPU @ 2.60Hz, 2592 Mhz, 4 Core(s), 8 Logical Processor(s) with 32GiB RAM.

Testing of the Static Analysis tools was conducted an Ubuntu 18.04.6 LTS installed on Dell Latitude E7250 with Processer Intel® Core™ i3-5010U CPU @ 2.10GHz, 1720Mhz, 4 Core(s), with 8GiB RAM.

Docker version 20.10.17, build 100c701, was installed on Ubuntu 18.04.6 LTS. Please note a departure from the installation instruction resulted in utilising “yarn” as an alternative to “npm” as the latter version experience difficulty in installation.

2 Dataset

2.1 Dataset

207 Smart Contracts were selected from two known sources, SWC Registry [1] and Smartbugs [2]. SWC registry, a Smart Contract Weakness classification registry hosted on GitHub under a MIT licence and maintained by smart contract developers, contains datasets of smart contracts written in solidity which have vulnerabilities and/or fixed vulnerabilities. The vulnerabilities are listed from SWC100 to SWC135 (at the time if writing) with commentary and remedies concerning the vulnerability provided. The registry is based on Common Weakness Enumeration CWE, a community-based list of software vulnerabilities. All 117 smart contracts form the registry will be tested.

The smart contracts extracted form Smartbugs are hosted on Smartbugs GitHub repository, a framework for analysing smart contracts. Contained within Smartbugs GitHub repository is a dataset of solidity smart contracts sourced for testing with accompanying comments on the location of the vulnerability within the contract. Unlike SWC, Smartbugs, lists the vulnerabilities into larger category groups. Smartbugs lists 9 separate groups, of which five relate to Solidity and four of the five were selected for testing due to the number of smart contracts per vulnerability, which included the following vulnerabilities, Re-entrancy, Access Control, Arithmetic and Unchecked Low-Level Checks, totalling 90 contracts. The contracts were sourced from Etherscan and other known vulnerable contracts.

3 Installation

3.1 VS Code

Install VS Code from https://code.visualstudio.com/docs?dv=win
Launch VS Code (Fig 1) and install extension Solidity for the writing and compiling of smart contracts written in Solidity (Fig 2).

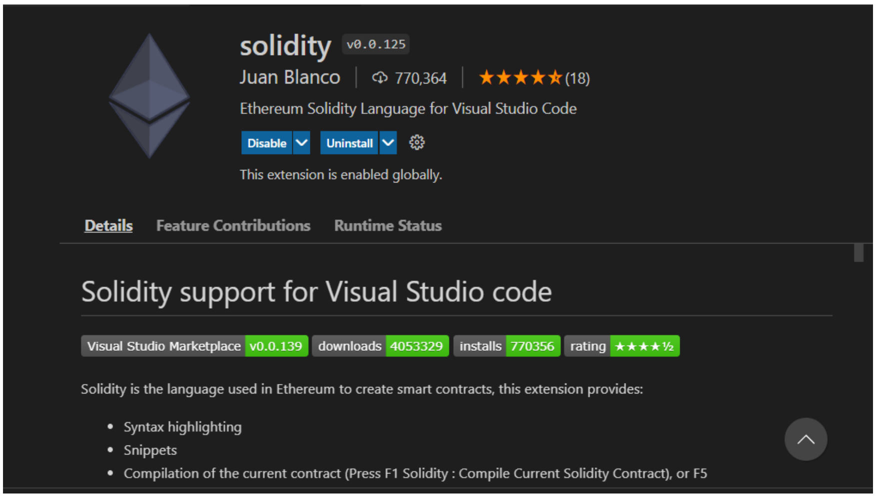


Fig 1: Solidity extension

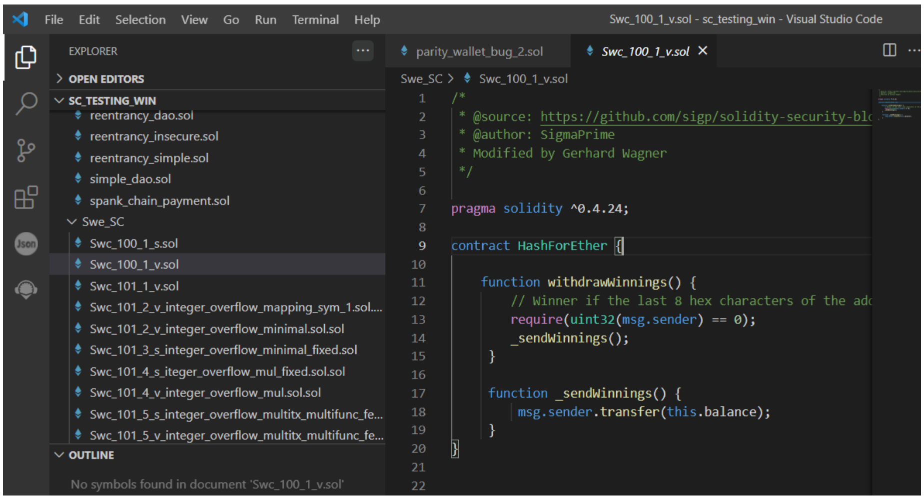


Fig 2: Writing smart contracts in VS Code

3.2 Docker Images Osiris, Oyente and Slither

Setup Docker:

Remove any existence of a previous version of docker:
```
$ sudo apt purge docker-desktop
```

Installing the Docker Community Engine:
```
$ sudo apt-get install ./docker-desktop-<version>-<arch>.deb
$ sudo apt-get install ca-certificates curl gnupg lsb-release
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
$ (lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

Detail of Docker can be seen in Fig 3.

```
bobby@bobby-Latitude-E7250:~$ systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2022-07-08 13:37:16 IST; 6h ago
     Docs: https://docs.docker.com
 Main PID: 2237 (dockerd)
    Tasks: 45
   CGroup: /system.slice/docker.service
           └─2237 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

Fig 3: Docker installed

Docker version (Fig 4)

```
bobby@bobby-Latitude-E7250:~$ docker --version
Docker version 20.10.17, build 100c701
bobby@bobby-Latitude-E7250:~$
```

Fig 4: Docker version

Pulling down images from docker hub https://hub.docker.com/
```
$ docker pull <ImageID>
```

Check for containers in Docker
```
$ docker ps -a
```

Search for the image ID of the test tool
```
$ docker image
```

Commence running image in docker. This will start a “container “which is a running image. The container allows for execution of commands and functionality. Run the image as a container in the background.
```
$ docker run -dit <ImageID>
```

Copy file containing smart contract into the container
```
$ docker cp <file> <ContianerID:<File>
```

Start container and execute commends from within the container
```
$ docker exec -it <ContainerID> /bin/bash
```

Docker images "pulled" were smartbugs/osiris [3], luongnguyen/oyente [4] and smartbugs/slither [5]. The version of Osiris , Oyente and Slither can be seen in Fig 5, Fig 6 and Fig 7.

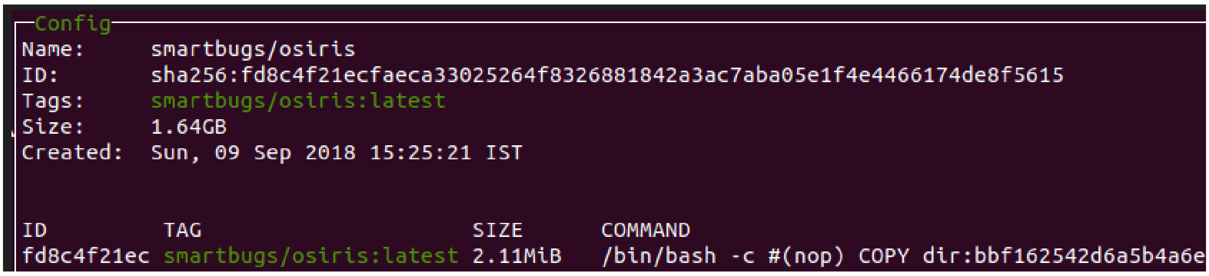


Fig 5: Osiris Image installed

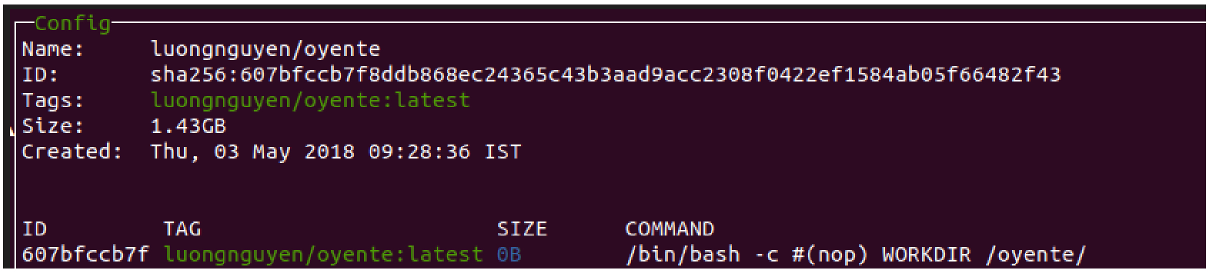


Fig 6: Oyente Image installed

Fig 7: Slither Image installed

3.3 Lazy Docker

Organise the docker images and containers in the terminal by installing Lazydocker (Fig 8). Version of lazydocker can be seen in Fig 9.

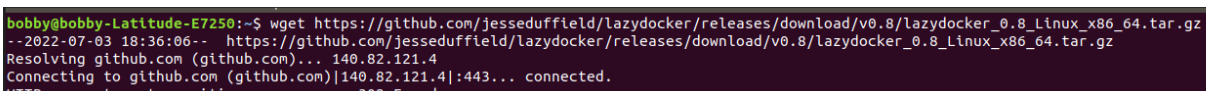


Fig 8: Installing Lazy Docker

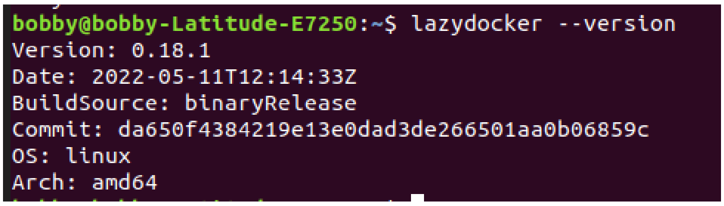

```
bobby@bobby-Latitude-E7250:~$ lazydocker --version
Version: 0.18.1
Date: 2022-05-11T12:14:33Z
BuildSource: binaryRelease
Commit: da650f4384219e13e0dad3de266501aa0b06859c
OS: linux
Arch: amd64
```

Fig 9: Lazy Docker version

Lazydocker is to organise the docker containers , however th eoperation of the containers can still be conducted through the terminal, which is the main process this experiment utilised/. Lazydocker was utilised as a quick reference to ascertina which containersn were running (Fig 10).

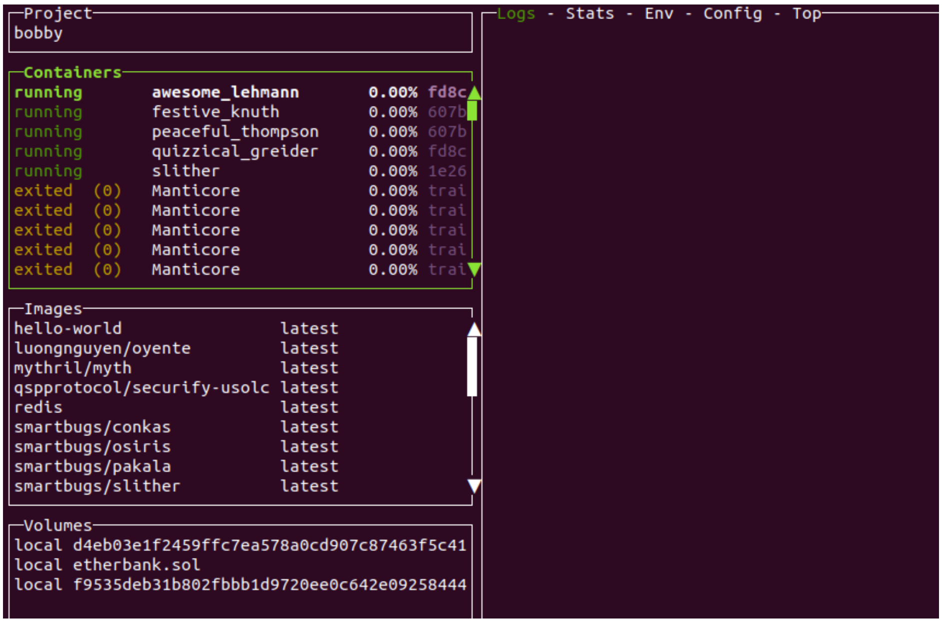


Fig 10: Lazydocker view of docker containers running

3.4 Execution of commands in Osiris, Oyente and Slither

From the terminal the smart contracts ae copied into the container as per instruction above. Start the container in an interactive shell requires the `docker exec` command as above. From within the container, commands are executed to run the test tool against the individual smart contracts.

Search for the image (Osiris)

```
$ docker image
```

Start the container with the image ID

```
$ docker run -dit <ImageID>
```

Search for the contain identification number

```
$ docker ps -a
```

With the identified container identification number execute a function to allow interaction with the container.

```
$ docker exec -it <ContainerID> /bin/bash
```

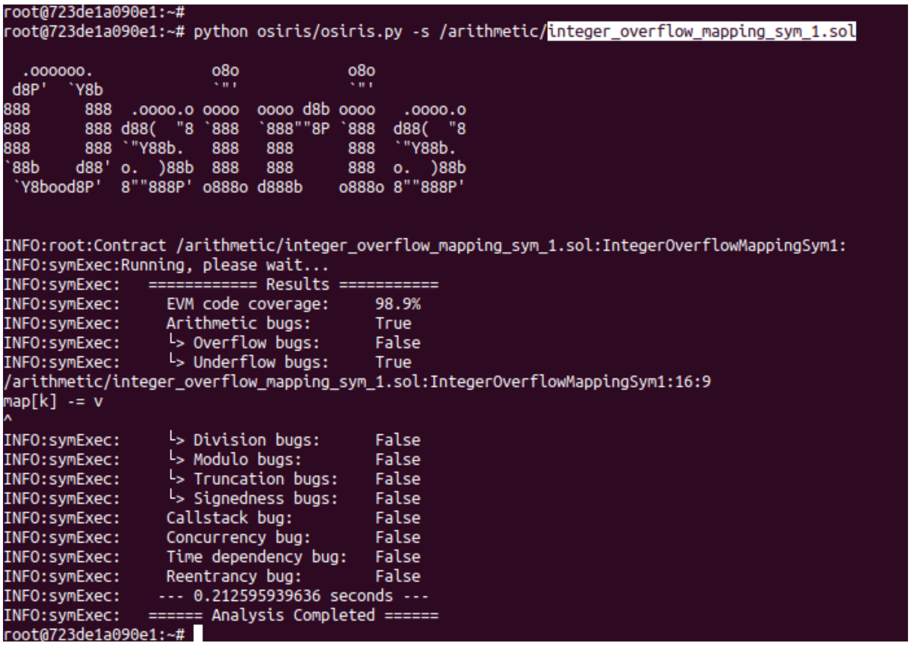
3.4.1 Commands in Osiris

From within the container:
`root@723de1a090e1: ~#`

Check folders in Osiris
`$ ls`

Utilizing the Osiris python script execute against the fill path of the location of the copied in smart contract (Fig 11):

```
root@723de1a090e1:~# python osiris/osiris.py -s
/arithmetic/integer_overflow_mapping_sym_1.solsol
```



```
root@723de1a090e1:~#
root@723de1a090e1:~# python osiris/osiris.py -s /arithmetic/integer_overflow_mapping_sym_1.sol

  .oooooo.             o8o            o8o
 d8P'  `Y8b            `"'            `"'
888      888  .oooo.o oooo  oooo d8b oooo   .oooo.o
888      888 d88(  "8 `888  `888""8P `888  d88(  "8
888      888 `"Y88b.   888   888      888  `"Y88b.
`88b    d88' o.  )88b  888   888      888  o.  )88b
 `Y8bood8P'  8""888P' o888o d888b    o888o 8""888P'


INFO:root:Contract /arithmetic/integer_overflow_mapping_sym_1.sol:IntegerOverflowMappingSym1:
INFO:symExec:Running, please wait...
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM code coverage:     98.9%
INFO:symExec:     Arithmetic bugs:       True
INFO:symExec:      └> Overflow bugs:      False
INFO:symExec:      └> Underflow bugs:     True
/arithmetic/integer_overflow_mapping_sym_1.sol:IntegerOverflowMappingSym1:16:9
map[k] -= v
^
INFO:symExec:      └> Division bugs:      False
INFO:symExec:      └> Modulo bugs:        False
INFO:symExec:      └> Truncation bugs:    False
INFO:symExec:      └> Signedness bugs:    False
INFO:symExec:     Callstack bug:         False
INFO:symExec:     Concurrency bug:       False
INFO:symExec:     Time dependency bug:   False
INFO:symExec:     Reentrancy bug:        False
INFO:symExec:    --- 0.212595939636 seconds ---
INFO:symExec:   ====== Analysis Completed ======
root@723de1a090e1:~#
```

Fig 11: Osiris result after execution of command

3.4.2 Commands in Oyente

From within the container:
`root@e4d5cb041e79:/oyente#`

Check folders in Oyente
`$ ls`

Navigate into the Oyente folder to execute the python script.

```
$ cd oyenter
root@e4d5cb041e79:/oyente/oyente#
```

Utilizing the Oyente python script execute against the fill path of the location of the smart in contract (Fig 12):

```
root@e4d5cb041e79:/oyente/oyente# python oyente.py -s
/reentrancy/reentrancy_insecure.sol
```

```
root@e4d5cb041e79:/oyente/oyente# python oyente.py -s /arithmetic/arithmetic/insecure_transfer.sol
WARNING:root:You are using evm version 1.8.2. The supported version is 1.7.3
WARNING:root:You are using solc version 0.4.21, The latest supported version is 0.4.19
INFO:root:contract /arithmetic/arithmetic/insecure_transfer.sol:IntegerOverflowAdd:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                     99.6%
INFO:symExec:     Integer Underflow:                     False
INFO:symExec:     Integer Overflow:                      True
INFO:symExec:     Parity Multisig Bug 2:                 False
INFO:symExec:     Callstack Depth Attack Vulnerability:  False
INFO:symExec:     Transaction-Ordering Dependence (TOD): False
INFO:symExec:     Timestamp Dependency:                  False
INFO:symExec:     Re-Entrancy Vulnerability:             False
INFO:symExec:/arithmetic/arithmetic/insecure_transfer.sol:18:9: Warning: Integer Overflow.
        balanceOf[_to] += _value
Integer Overflow occurs if:
    _value = 44369063854674067291029404066660873444229566625561754964912869797988903417852
    balanceOf[_to] = 85653202831209899131921273706816539903532775246499202405936884825549521553152
    balanceOf[msg.sender] = 44369063854674067291029404066660873444229566625561754964912869797988903417852
INFO:symExec:   ====== Analysis Completed ======
```

Fig 12: Oyente result after execution of command

3.4.3 Commands in Slither

Commands for Slither:
From within the container:
`root@8439351fd412:/slither#`

Command execute against a smart contract (Fig 13)
`root@8439351fd412:/slither# slither /unchecked_low_level_calls/0x7a4349a749e59a5736efb7826ee3496a2dfd5489.sol`

```
INFO:Detectors:
Reentrancy in PrivateBank.CashOut (/reentrancy/0xb93430ce38ac4a6bb47fb1fc085ea669353fd89e.sol#34-47):
        External calls:
        - msg.sender.call.value(_am)() (/reentrancy/0xb93430ce38ac4a6bb47fb1fc085ea669353fd89e.sol#39-46)
        State variables written after the call(s):
        - balances (/reentrancy/0xb93430ce38ac4a6bb47fb1fc085ea669353fd89e.sol#41)
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#reentrancy-vulnerabilities
```

Fig 13: Partial return from Slither after execution of command

4 Vulnerabilities tested by tools

4.1 Osiris

The following are Solidity smart contract vulnerabilities detected by Osiris as determined by previous studies [6] [7].

- Assertion failure
- State Dependency
- Integer Overflow/Underflow
- Denial of Service
- Time Manipulation
- Re-entrancy

4.2 Oyente

The following are Solidity smart contract vulnerabilities detected by Oyente as determined by previous studies [6] [8] [9] [10] [11] [7] [12].

- Re-entrancy
- Unhandled Exceptions
- Transaction Order dependency
- Integer Overflow/underflow
- Timestamp dependency
- Tx. Order Dependence
- State Dependence
- Assertion failure
- Freezing ether
- Denial of Service
- Time Manipulation

4.3 Slither

The following are document (Fig 14) Solidity smart contract vulnerabilities detectable by Slither [13].

Num	Detector	What it Detects	Impact	Confidence
1	`shadowing-state`	State variables shadowing	High	High
2	`suicidal`	Functions allowing anyone to destruct the contract	High	High
3	`uninitialized-state`	Uninitialized state variables	High	High
4	`uninitialized-storage`	Uninitialized storage variables	High	High
5	`arbitrary-send`	Functions that send ether to arbitrary destinations	High	Medium
6	`controlled-delegatecall`	Controlled delegatecall destination	High	Medium
7	`reentrancy-eth`	Reentrancy vulnerabilities (theft of ethers)	High	Medium
8	`erc20-interface`	Incorrect ERC20 interfaces	Medium	High
9	`incorrect-equality`	Dangerous strict equalities	Medium	High
10	`locked-ether`	Contracts that lock ether	Medium	High
11	`shadowing-abstract`	State variables shadowing from abstract contracts	Medium	High
12	`constant-function`	Constant functions changing the state	Medium	Medium
13	`reentrancy-no-eth`	Reentrancy vulnerabilities (no theft of ethers)	Medium	Medium
14	`tx-origin`	Dangerous usage of `tx.origin`	Medium	Medium
15	`uninitialized-local`	Uninitialized local variables	Medium	Medium
16	`unused-return`	Unused return values	Medium	Medium
17	`shadowing-builtin`	Built-in symbol shadowing	Low	High
18	`shadowing-local`	Local variables shadowing	Low	High
19	`calls-loop`	Multiple calls in a loop	Low	Medium
20	`reentrancy-benign`	Benign reentrancy vulnerabilities	Low	Medium
21	`timestamp`	Dangerous usage of `block.timestamp`	Low	Medium
22	`assembly`	Assembly usage	Informational	High
23	`constable-states`	State variables that could be declared constant	Informational	High
24	`deprecated-standards`	Deprecated Solidity Standards	Informational	High
25	`erc20-indexed`	Un-indexed ERC20 event parameters	Informational	High
26	`external-function`	Public function that could be declared as external	Informational	High
27	`low-level-calls`	Low level calls	Informational	High
28	`naming-convention`	Conformance to Solidity naming conventions	Informational	High
29	`pragma`	If different pragma directives are used	Informational	High
30	`solc-version`	Incorrect Solidity version (< 0.4.24 or complex pragma)	Informational	High
31	`unused-state`	Unused state variables	Informational	High

Fig 14: Vulnerablities detected by Slither [13]

5 Testing

5.1 Implementation Testing of Tools and Dataset

The Static Analysis tools from Solidity based Smart Contracts were sourced form Docker Hub. Each smart tool was a docker image.
Install docker Community Edition Engine from the repository as guided from the Docker Documents. Pulling the Selected Static Analytic Tools docker images, Oyente, Slither and Osiris. Using docker execute commands run the images as container.

To copy the smart contract, run docker detect mode "docker run -dit <image_name> /bin/bash". This command allows the container to run in the background so the files can be copied into the container.

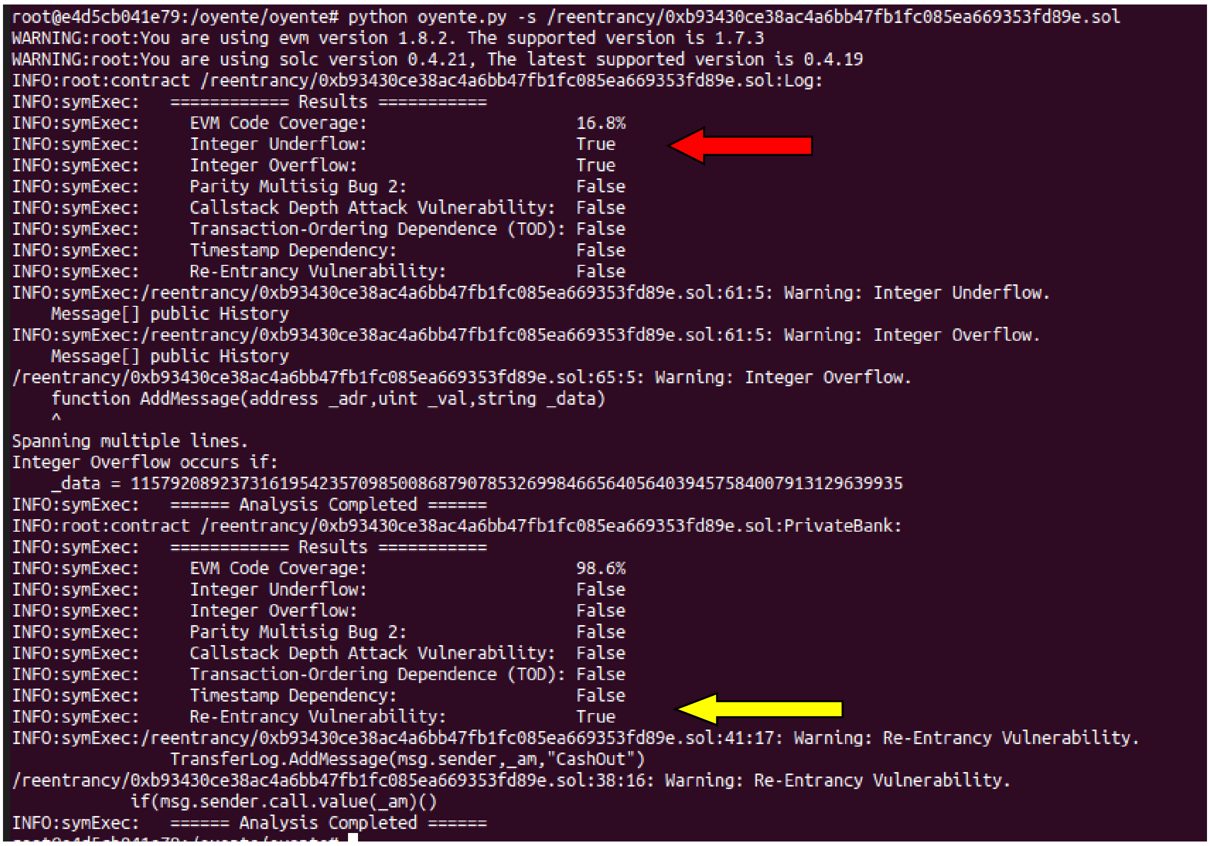
After the copying of the smart contracts to the test tool docker containers, using terminal commands in docker, the smart contracts are individual tested. The result comprises of the written findings from the tools onto a terminal screen. Each tool conducts the examination differently, and the results are output according to each tool as illustrated in Fig 15, showing a true Positive result for re-entrancy vulnerability.

```
INFO:root:contract /reentrancy/0x941d225236464a25eb18076df7da6a91d0f95e9e.sol:ETH_FUND:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                     98.6%
INFO:symExec:     Integer Underflow:                     False
INFO:symExec:     Integer Overflow:                      False
INFO:symExec:     Parity Multisig Bug 2:                 False
INFO:symExec:     Callstack Depth Attack Vulnerability:  False
INFO:symExec:     Transaction-Ordering Dependence (TOD): False
INFO:symExec:     Timestamp Dependency:                  False
INFO:symExec:     Re-Entrancy Vulnerability:             True
INFO:symExec:/reentrancy/0x941d225236464a25eb18076df7da6a91d0f95e9e.sol:44:16: Warning: Re-Entrancy Vulnerability.
            if(msg.sender.call.value(_am)()
/reentrancy/0x941d225236464a25eb18076df7da6a91d0f95e9e.sol:47:17: Warning: Re-Entrancy Vulnerability.
                TransferLog.AddMessage(msg.sender,_am,"CashOut")
INFO:symExec:   ====== Analysis Completed ======
```

Fig 15: Oytene Output sample

The above is a successful detection for the vulnerability re-Entrancy. This is atypical of a result. However, some results can be more extensive with more output as the program iterates through the smart contract.

However, results can be hybrid returns a seen below in Fig 16, Fig 17 and Fig 18.
The example below, the smart contract is listed with a re-entrancy vulnerability. Oyente and Slither detect the re-entrancy vulnerability. Osiris failed to detect for re-entrancy. However, both Oyente and Osiris detected Arithmetic Vulnerability within the re-entrancy vulnerable smart contract which was not listed as a vulnerability.



```
root@e4d5cb041e79:/oyente/oyente# python oyente.py -s /reentrancy/0xb93430ce38ac4a6bb47fb1fc085ea669353fd89e.sol
WARNING:root:You are using evm version 1.8.2. The supported version is 1.7.3
WARNING:root:You are using solc version 0.4.21, The latest supported version is 0.4.19
INFO:root:contract /reentrancy/0xb93430ce38ac4a6bb47fb1fc085ea669353fd89e.sol:Log:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                     16.8%
INFO:symExec:     Integer Underflow:                     True
INFO:symExec:     Integer Overflow:                      True
INFO:symExec:     Parity Multisig Bug 2:                 False
INFO:symExec:     Callstack Depth Attack Vulnerability:  False
INFO:symExec:     Transaction-Ordering Dependence (TOD): False
INFO:symExec:     Timestamp Dependency:                  False
INFO:symExec:     Re-Entrancy Vulnerability:             False
INFO:symExec:/reentrancy/0xb93430ce38ac4a6bb47fb1fc085ea669353fd89e.sol:61:5: Warning: Integer Underflow.
    Message[] public History
INFO:symExec:/reentrancy/0xb93430ce38ac4a6bb47fb1fc085ea669353fd89e.sol:61:5: Warning: Integer Overflow.
    Message[] public History
/reentrancy/0xb93430ce38ac4a6bb47fb1fc085ea669353fd89e.sol:65:5: Warning: Integer Overflow.
    function AddMessage(address _adr,uint _val,string _data)
    ^
Spanning multiple lines.
Integer Overflow occurs if:
    _data = 115792089237316195423570985008687907853269984665640564039457584007913129639935
INFO:symExec:   ====== Analysis Completed ======
INFO:root:contract /reentrancy/0xb93430ce38ac4a6bb47fb1fc085ea669353fd89e.sol:PrivateBank:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                     98.6%
INFO:symExec:     Integer Underflow:                     False
INFO:symExec:     Integer Overflow:                      False
INFO:symExec:     Parity Multisig Bug 2:                 False
INFO:symExec:     Callstack Depth Attack Vulnerability:  False
INFO:symExec:     Transaction-Ordering Dependence (TOD): False
INFO:symExec:     Timestamp Dependency:                  False
INFO:symExec:     Re-Entrancy Vulnerability:             True
INFO:symExec:/reentrancy/0xb93430ce38ac4a6bb47fb1fc085ea669353fd89e.sol:41:17: Warning: Re-Entrancy Vulnerability.
                TransferLog.AddMessage(msg.sender,_am,"CashOut")
/reentrancy/0xb93430ce38ac4a6bb47fb1fc085ea669353fd89e.sol:38:16: Warning: Re-Entrancy Vulnerability.
            if(msg.sender.call.value(_am)()
INFO:symExec:   ====== Analysis Completed ======
```

Fig 16: Oyente Output Re-entrancy Vulnerability

Fig 16 shows a TP for re-entrancy and a detection of Arithmetic vulnerability. A process is actioned to check if the Arithmetic detection is TP or FP (see below).

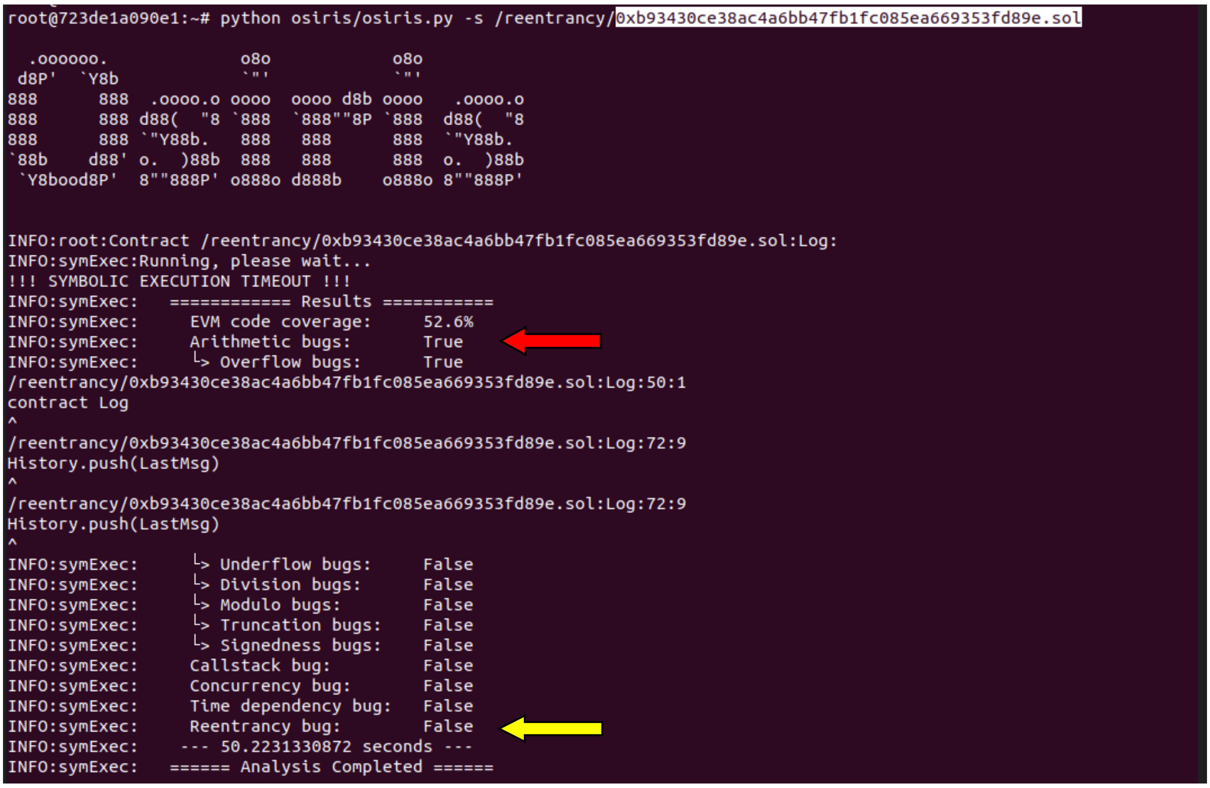


Fig 17: Osiris Output Re-entrancy Vulnerability

Fig 17 above shows the failed detection of re-entrancy but postive detection for an arithmetic vulnerablity. The test gives a FN for re-entracy but a detection for arithmic. It needs to be determined if the detecion is a TP of a FP. Examiantion of both Fig 16 and Fig 17 shows Oyente also detected to unlisted vulenrablity, thereby both Oyente and Osiris are TP for detecting this arithmetic vulnerablity. The TP for Arithmetic will be included intot he Arithmetic Vulnerablity dataset generated.

Slither Output sample:

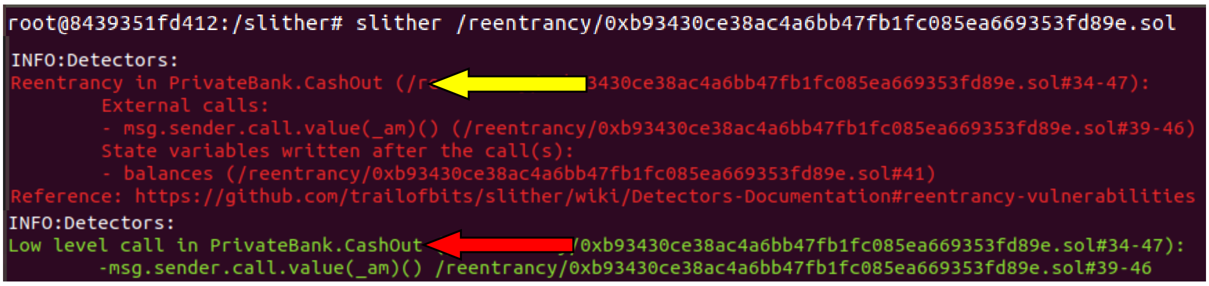


Fig 18: Slither Output Re-entrancy Vulnerability

(Fig 18 image was cropped for illustration purposes)
Diagram 18 illustrates the Slither tool detection of re-entrancy, however, there is a detection for Low Level Call. The detection for Low Level Call (LLC) is within the tools scope to detect for this vulnerability; however, this vulnerability was not listed as a vulnerability for this smart contract. Further research on the vulnerability and contract is conducted, however, it could not be determined that the smart contract contained the LLC vulnerability, therefore the detection by Slither of Low-Level Call will be consider FP, for the purpose of the experiment.

6 Metrics

Recall, Precision, Accuracy and F1-Score were calculated for each testing tool.

Recall = TP / (TP + FN) How many relevant items were observed? Related to a type II error.
Precision = TP / (TP + FP) How many observed items are relevant? Related to a type I error.
Accuracy = (TP +TN) / (TP + TN + FP + FN) Accuracy represents the number of correctly classified data observations over the total number of observations.
The F1 Score: = 2* (Recall*Precision) / (Recall + Precision)

Results are entered into spread sheet with smart contracts and corresponding TP,TN , FP and FN (Fig 19)

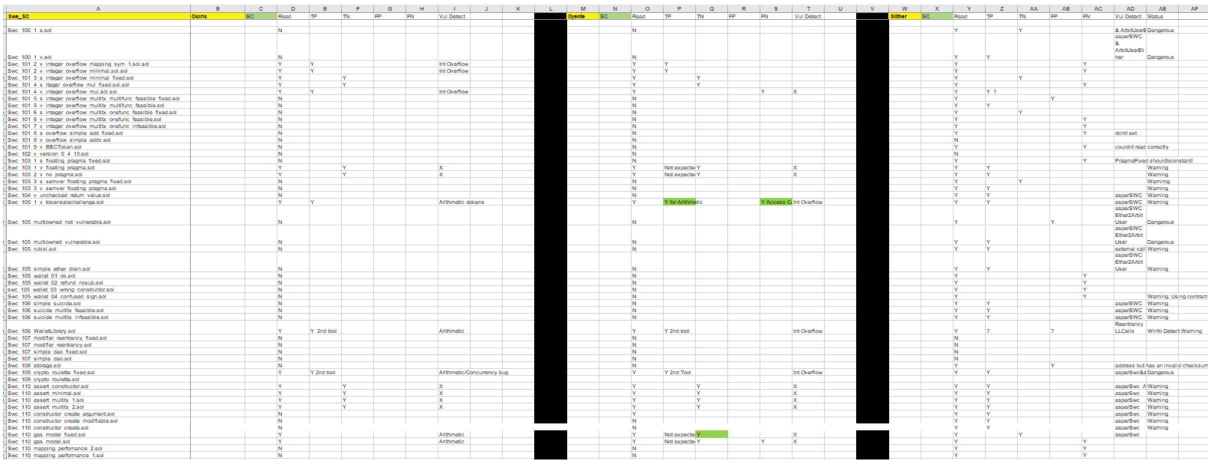
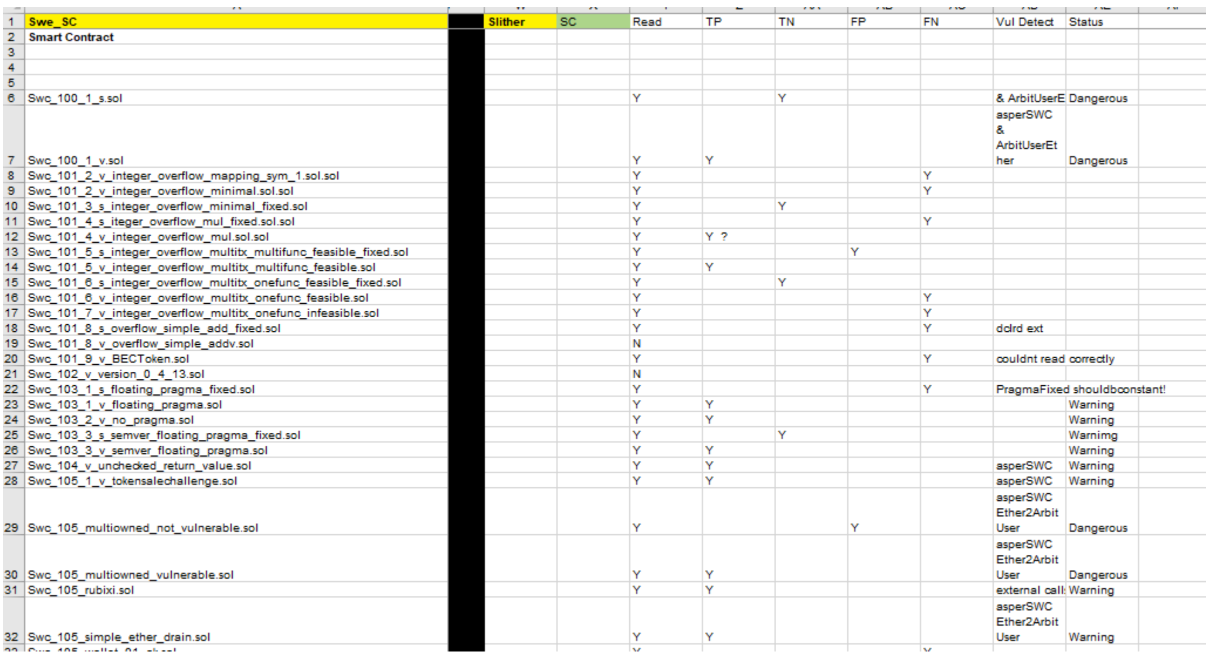


Fig 19: Results of TP, TN, FP and FN entered into Spread Sheet.



	Swe_SC	Slither	SC	Read	TP	TN	FP	FN	Vul Detect	Status
2	Smart Contract									
3										
4										
5										
6	Swc_100_1_s.sol			Y		Y			& ArbitUserE	Dangerous
7	Swc_100_1_v.sol			Y	Y				asperSWC & ArbitUserEt her	Dangerous
8	Swc_101_2_v_integer_overflow_mapping_sym_1.sol.sol			Y				Y		
9	Swc_101_2_v_integer_overflow_minimal.sol.sol			Y				Y		
10	Swc_101_3_s_integer_overflow_minimal_fixed.sol			Y		Y				
11	Swc_101_4_s_iteger_overflow_mul_fixed.sol.sol			Y				Y		
12	Swc_101_4_v_integer_overflow_mul.sol.sol			Y	Y ?					
13	Swc_101_5_s_integer_overflow_multitx_multifunc_feasible_fixed.sol			Y			Y			
14	Swc_101_5_v_integer_overflow_multitx_multifunc_feasible.sol			Y	Y					
15	Swc_101_6_s_integer_overflow_multitx_onefunc_feasible_fixed.sol			Y		Y				
16	Swc_101_6_v_integer_overflow_multitx_onefunc_feasible.sol			Y				Y		
17	Swc_101_7_v_integer_overflow_multitx_onefunc_infeasible.sol			Y				Y		
18	Swc_101_8_s_overflow_simple_add_fixed.sol			Y				Y	dclrd ext	
19	Swc_101_8_v_overflow_simple_addv.sol			N						
20	Swc_101_9_v_BECToken.sol			Y				Y	couldnt read correctly	
21	Swc_102_v_version_0_4_13.sol			N						
22	Swc_103_1_s_floating_pragma_fixed.sol			Y				Y	PragmaFixed shouldbconstant!	
23	Swc_103_1_v_floating_pragma.sol			Y	Y					Warning
24	Swc_103_2_v_no_pragma.sol			Y	Y					Warning
25	Swc_103_3_s_semver_floating_pragma_fixed.sol			Y		Y				Warnimg
26	Swc_103_3_v_semver_floating_pragma.sol			Y	Y					Warning
27	Swc_104_v_unchecked_return_value.sol			Y	Y				asperSWC	Warning
28	Swc_105_1_v_tokensalechallenge.sol			Y	Y				asperSWC	Warning
29	Swc_105_multiowned_not_vulnerable.sol			Y			Y		asperSWC Ether2Arbit User	Dangerous
30	Swc_105_multiowned_vulnerable.sol			Y	Y				asperSWC Ether2Arbit User	Dangerous
31	Swc_105_rubixi.sol			Y	Y				external call:	Warning
32	Swc_105_simple_ether_drain.sol			Y	Y				asperSWC Ether2Arbit User	Warning

Fig 20: Enlarge sample of Spread Sheet from Fig19

References

[1] 'SWC-100 · Overview'. http://swcregistry.io/ (accessed Jul. 10, 2022).

[2] 'SmartBugs: A Dataset of Vulnerable Solidity Smart Contracts'. https://smartbugs.github.io/ (accessed Jun. 04, 2022).

[3] 'smartbugs/osiris - Docker Image | Docker Hub'. https://hub.docker.com/r/smartbugs/osiris (accessed Aug. 14, 2022).

[4] 'luongnguyen/oyente - Docker Image | Docker Hub'. https://hub.docker.com/r/luongnguyen/oyente (accessed Aug. 14, 2022).

[5] 'smartbugs/slither - Docker Image | Docker Hub'. https://hub.docker.com/r/smartbugs/slither (accessed Aug. 14, 2022).

[6] J. Choi, D. Kim, S. Kim, G. Grieco, A. Groce, and S. K. Cha, 'SMARTIAN: Enhancing Smart Contract Fuzzing with Static and Dynamic Data-Flow Analyses', in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov. 2021, pp. 227–239. doi: 10.1109/ASE51524.2021.9678888.

[7] M. Ren *et al.*, 'Empirical evaluation of smart contract testing: what is the best choice?', in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, New York, NY, USA, Jul. 2021, pp. 566–579. doi: 10.1145/3460319.3464837.

[8] A. Ghaleb and K. Pattabiraman, 'How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection', in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, Virtual Event USA, Jul. 2020, pp. 415–427. doi: 10.1145/3395363.3397385.

[9] H. Chen, M. Pendleton, L. Njilla, and S. Xu, 'A Survey on Ethereum Systems Security: Vulnerabilities, Attacks, and Defenses', *ACM Comput. Surv.*, vol. 53, no. 3, pp. 1–43, May 2021, doi: 10.1145/3391195.

[10] M. Zhang, X. Zhang, Y. Zhang, and Z. Lin, 'TXSPECTOR: Uncovering Attacks in Ethereum from Transactions', p. 19, Aug. 2020.

[11] D. Perez and B. Livshits, 'Smart Contract Vulnerabilities: Vulnerable Does Not Imply Exploited', 2021, pp. 1325–1341. Accessed: May 01, 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/perez

[12] C. Benabbou and Ö. Gürcan, 'A Survey of Verification, Validation and Testing Solutions for Smart Contracts', in *2021 Third International Conference on Blockchain Computing and Applications (BCCA)*, Nov. 2021, pp. 57–64. doi: 10.1109/BCCA53669.2021.9657040.

[13] 'trailofbits/slither - Docker Image | Docker Hub'. https://hub.docker.com/r/trailofbits/slither (accessed Aug. 01, 2022).